ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет безопасности информационных технологий

Кафедра проектирования и безопасности компьютерных систем

Дисциплина:

«Операционные системы»

ОТЧЕТ О ВЫПОЛНЕНИИ

ЛАБОРАТОРНОЙ РАБОТЫ №9

(Сложный вариант, модуль ядра и интерфейс netfilter)

Выполнил:

Студент гр. N3247 Гаврилова В.В.

Проверил:

Ханов А. Р.

Санкт-Петербург

2021г.

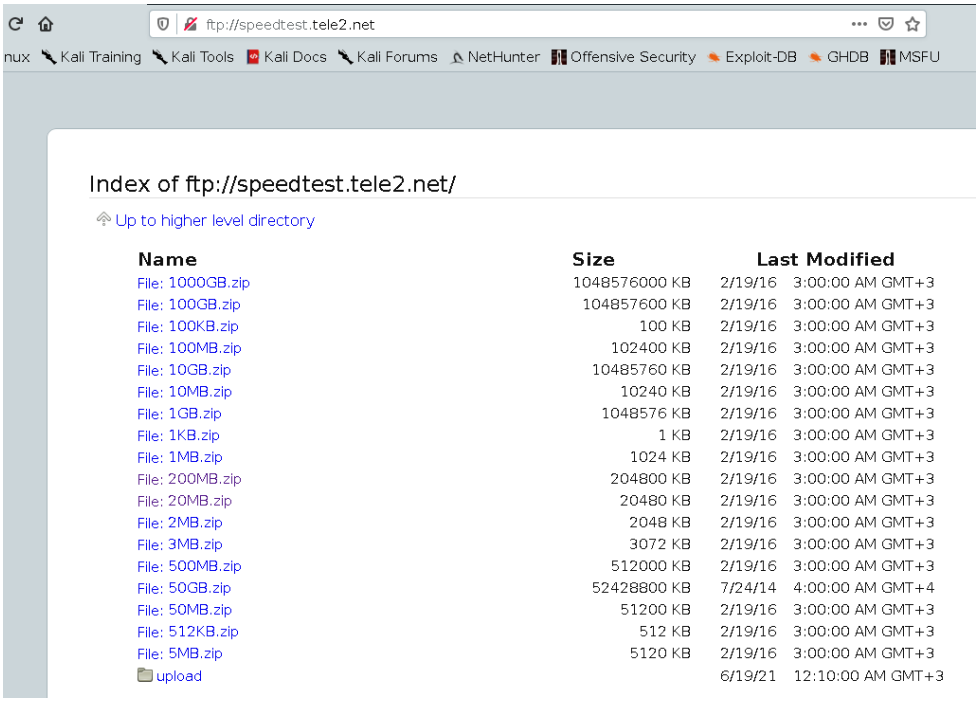
Постановка задачи: найти потери времени при передаче пакетов, которая происходит при обработке пакетов netfilter.

Решено сделать следующее: давайте при помощи тестового сервиса ftp://speedtest.tele2.net будем скачивать архив произвольного объема (в нашем случае 20 Мб) соответственно используя протокол ftp.

В самом деле, в пакете нас будут интересовать следующие вещи: ethernet заголовок, так как пакет у нас обслуживается протоколами ethernet канального уровня (остальное будем просто пропускать), ip заголовок сетевого уровня (здесь нас интересует адрес отправителя пакета 90.130.70.73), tcp заголовок транспортного уровня (тут нас будут интересовать порт отправителя и флаги, выставленные на пакете). В более общих случаях нас могут интересовать конкретные данные (пометки, информация в пакете) уже соответствующие самому ftp, в таком случае мы могли бы читать еще и саму data пакета (sk_buff структура из ядра, в которой хранятся и при помощи которой обрабатываются пакеты, не дифференцирует протоколы более высокого уровня, поэтому эти данные приходится размечать и читать вручную). В нашем случае это не нужно.

Суть следующая: мы как клиент обращаемся к 21 порту ftp сервера, таким образом осуществляем запросы к нему, а вот сами данные сервер передает нам переходя в PASV мод, то есть открывает произвольный порт с большим значением и с него осуществляет передачу пакетов запрошенных данных. Наш фильтр пакетов будет регистрировать хук на этапе PRE_ROUTING (до маршрутизации, то есть как только пакет попадает к нам) с самым высоким приоритетом, который будет помещать пакет в структуру sk_buff, в котором мы будем смотреть адрес и порт сервера-отправителя, а отсчитывать общее время и замерять будем пакеты которые пришли НЕ с 21го порта. До этого, само собой, трафик был исследован wireshark'ом, поэтому мы точно знаем, что с ftp сервера мы получаем данные только с 21го порта (служебные) и с любого кроме 21го (целевые). Заканчивать замеры будем по концу передачи, то есть когда с НЕ 21го порта придет пакет с флагом FIN.

Качать будем 20Мб пустой архив.



Код модуля ядра:

```c
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/types.h>
#include <linux/time64.h>
#include <linux/timekeeping.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/skbuff.h>
#include <linux/ip.h>
#include <linux/tcp.h>


unsigned long long int tim(void){
    struct timespec64 t;
    ktime_get_ts64(&t);
    return t.tv_sec*1000000000LL + t.tv_nsec;
}

static struct nf_hook_ops nfho;
static unsigned long long int pack_cnt;
static unsigned long long int tot_trans_tim;
static unsigned long long int packet_proc_sum_tim;
static bool tr_flag = false;

unsigned int hook_func(void *priv, struct sk_buff *skb, const struct nf_hook_state *state){

    printk(KERN_INFO "nfter: executing hook\n");

    unsigned long long int t1 = tim();
    struct iphdr *ip;
    struct tcphdr *tcp;
    __be32 src_addr = ntohl(0x5A824649); // src addr = 90.130.70.73 ftp://speedtest.tele2.net

    if (skb->protocol == htons(ETH_P_IP)){
        printk(KERN_INFO "nfter: got eth header of packet\n");
        ip = (struct iphdr *)skb_network_header(skb);
        if (ip->version == 4 && ip->protocol == IPPROTO_TCP && ip->saddr == src_addr){
            printk(KERN_INFO "nfter: got ip header of target packet \n");
            skb_set_transport_header(skb, ip->ihl * 4);
            tcp = (struct tcphdr *)skb_transport_header(skb);
            if (tcp->source == htons(21)){
                unsigned long long int t2;
                if (tcp->psh && tcp->ack){
                    if (!tr_flag){
                        tr_flag = true;
                        tot_trans_tim = tim();
                        pack_cnt = 0;
                        packet_proc_sum_tim = 0;
                        printk(KERN_INFO "nfter: TRANSMISSION START\n");
                    }
                    pack_cnt++;
                    printk(KERN_INFO "nfter: gettin packet numba %llu\n", pack_cnt);
                    t2 = tim() - t1;
                    packet_proc_sum_tim += t2;
```

```
                    if (!tr_flag){
                        tr_flag = true;
                        tot_trans_tim = tim();
                        pack_cnt = 0;
                        packet_proc_sum_tim = 0;
                        printk(KERN_INFO "nfter: TRANSMISSION START\n");
                    }
                    pack_cnt++;
                    printk(KERN_INFO "nfter: gettin packet numba %llu\n", pack_cnt);
                    t2 = tim() - t1;
                    packet_proc_sum_tim += t2;
                }
                if (tcp->fin || tcp->rst){
                    t2 = tim() - tot_trans_tim;
                    printk(KERN_INFO "nfter: TRANSMISSION ENDED; \nstats: total time -
                    %llu ns\n\t avg. time per packet (nfter) - %llu ns\n\t total time loss
                    %llu ns ;\n", t2, packet_proc_sum_tim/pack_cnt, packet_proc_sum_tim);
                }
                return NF_ACCEPT;
            }
            else return NF_ACCEPT;
        }
        else return NF_ACCEPT;
    }
    //else printk(KERN_INFO "nfter: not eth??\n");
    return NF_ACCEPT;

}


static int __init nfter_init(void){

    printk(KERN_INFO "nfter: loading module\n");
    nfho.hook = hook_func;
    nfho.hooknum = NF_INET_PRE_ROUTING;
    nfho.pf = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &nfho);

    return 0;
}

static void __exit nfter_exit(void){
    nf_unregister_net_hook(&init_net, &nfho);
    printk(KERN_INFO "nfter: goodbye\n");
}

module_init(nfter_init);
module_exit(nfter_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("w4r3z $c3n3r$ky");
MODULE_DESCRIPTION("simply catching packets in kernel spaaaace");
```

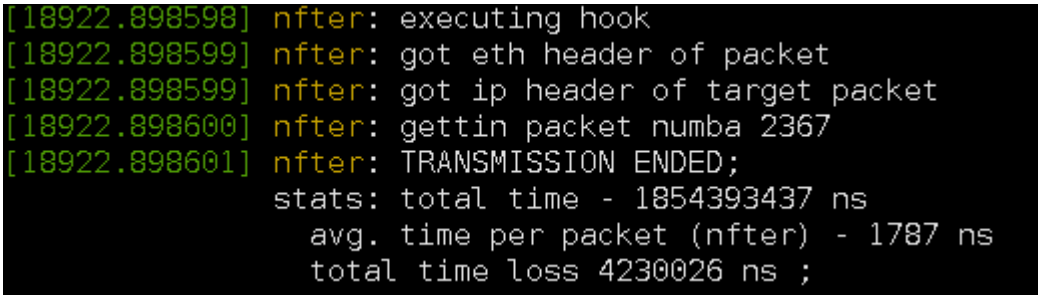
Собираем модуль (пришлось откатиться с кастомного ядра 5.10.24 до 5.8.2, на кастомное модуль не вставал):

```
root@kali:/mnt/hgfs/vm_share/labs/OS/lab9# cat Makefile
obj-m += nfter.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

make && insmod nfter.ko

Качаем архив

rmmod nfter.ko



Таки образом нам удалось оценить общее время, затраченное на передачу архива (~1,85 сек), оценить среднее время на обработку пакета фильтром и общее потерянное при передаче время (меньше одного процента).

Хотя, к такому тесту есть и замечания, потому как, например, в наших условия отсутствовал интенсивный входящий и исходящий трафик, а также это по факту был единственный хук, фильтр пакетов не большой и не развитой. Думаю, при большом количестве параллельно срабатывающих хуков и сложной маршрутизации эта доля потерь может оказать существенный отрицательный эффект на скорость обмена данными между узлами сети.